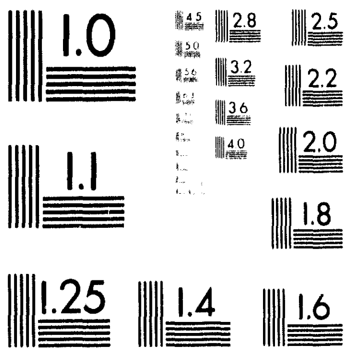

1.0
4.5
5.0
5.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

2

Conf-931160--19

An Efficient Communication Scheme for Solving the S_n Equations on Message-Passing Multiprocessors[*]

Yousry Y. Azmy
Engineering Physics and Mathematics Division
Oak Ridge National Laboratory
P.O. Box 2008, Bldg. 6025
Oak Ridge, Tennessee 37831-6363

Paper to be presented at the *American Nuclear Society 1993 Winter Meeting*, November 14-19, 1993, San Francisco, California.

*Research sponsored by the U.S. Department of Energy, managed by Martin Marietta Energy Systems, Inc., under contract No. DE-AC05-84OR21400.

AN EFFICIENT COMMUNICATION SCHEME FOR SOLVING THE S_n EQUATIONS ON MESSAGE-PASSING MULTIPROCESSORS

Y. Y. Azmy
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831

Early models of Intel's hypercube multiprocessors, e.g. the iPSC/1 and iPSC/2, were characterized by the high latency of message-passing. This relatively weak dependence of the communication penalty on the size of messages, in contrast to its strong dependence on the number of messages, justified using the *Fan-in Fan-out* algorithm (which implements a minimum spanning tree path) to perform global operations, such as global sums, etc. Recent models of message passing computers, such as the iPSC/860 and the Paragon, have been found to possess much smaller latency,[1] thus forcing a re-examination of the issue of performance optimization with respect to communication schemes.[2] Essentially, the Fan-in Fan-out scheme minimizes the number of nonsimultaneous messages sent but not the volume of data traffic across the network. Furthermore, if a global operation is performed in conjunction with the message-passing, a large fraction of the attached nodes remains idle as the number of utilized processors is halved in each step of the process. On the other hand, the Recursive Halving scheme offers the smallest communication cost for global operations,[2] but has some drawbacks. First, it requires the simultaneous exchange of messages between adjacent nodes, which while permissible on many message-passing computers, requires additional programing on the iPSC/860, the target platform in this work. Second, full utilization of the processors requires that the message length be a multiple of two, resulting in significant idleness of the processors if this is not the case. In this paper we present an alternative scheme that eliminates the first drawback by communicating along a mono-directional ring, and reduces the impact of the second drawback by requiring only that the number of nodes divides the message length, a standard requirement for retaining load balance.

In the *Bucket* scheme each node p is directly connected to adjacent nodes $p_+ = G(G^{-1}(p)+1)$ and $p_- = G(G^{-1}(p)-1)$, where G is the Gray code sequence function. The distributed vector on each processor is divided into P nonintersecting subvectors $V_{p,q}$, $q=1,...,P$, each of length V/P. The *global combine operation* is composed of a combine stage, and a broadcast stage, each performed in $P-1$ steps. In step $n = 1,...,P-1$ of the combine stage each node p sends V_{p,q^n} to node p_+, recieves V_{p_-,q^n} into buffer U_p and combines it with $V_{p,q^{n+1}}$, where $q^{n+1} = G(G^{-1}(q^n)-1)$, $n > 1$, and $q^1 = p$. Note that in the above q^n implicitly depends on p. At the conclusion of the combine stage each processor possesses the part of the final result stored in subvector $V_{p,p-1}$. The broadcast stage follows the same path as described above whereby each node sends the final result to p_+ and recieves it from p_- recursively, until each node possesses the final result in its entire local vector V_p.

Based on the above description of the Bucket scheme, the execution time for performing a global combine operation on P processors can be modeled by,

$$T_r(V,P) = (P-1)\ [2\tau_0 + \frac{V}{P}\ (2\ \tau_1 + \tau_o)], \qquad (1)$$

where τ_i, $i=0,1,o$ are constants representing communication latency, volumetric communication rate, and the combine operation execution time, respectivley. This is a significant improvement over the Fan-in Fan-out scheme which requires,[2]

$$T_t(V,P) = \log_2 P\ \ [2\tau_0 + V(2\tau_1 + \tau_o)], \qquad (2)$$

for $V \gg P$ large.

To demonstrate the Bucket scheme described above, we implement it in the two-dimensional Cartesian-geometry, Parallel-General Order Neutron Transport code *P-GONT*, originally implemented on the iPSC/2 using a Fan-in Fan-out scheme.[3] This new implementation is based on a nonsimultaneous decomposition of the angle, and space and method-order domains. As before,[3] the mesh sweeps are performed concurrently via the angle-domain decomposition, but the global sum used to construct the scalar flux from the various processors' angular flux contributions here is performed via the Bucket scheme. Furthermore, the convergence test that was previously performed on all processors simultaneously due to the high communication penalty,[3] here is performed concurrently immediately following the conclusion of the combine stage. During the broadcast stage, each processor sends the relevent final subvector amended with the maximum relative iteration residue, so that at the end of the broadcast stage each node can test the latter quantity against the convergence criterion, and determine whether or not to terminate the iterations. Hence the performance of the resulting code improves for two reasons. First, the better efficiency of the Bucket scheme; second, the serial component is reduced to the problem setup time only, which in most cases is negligible compared to the total execution time.

Next we construct and validate a performance model for the new scheme on the iPSC/860 hypercube at ORNL along the same lines detailed in Ref. 3. Indeed the mathematical model for the execution time is but a slight modification of the Fan-in Fan-out model,[3] wherein the global operation component is replaced by Eq. (1). We evaluate the model parameters using two S_8,16×16, and 32×32 mesh simple test problems, then we verify sperately the serial, parallel, and global components of the model against actual measured values for a third S_{16},32×32 mesh problem, and observe very good agreement.

Finally, we use the performance models to predict the parallel efficiency for the

Bucket and Fan-in Fan-out schemes for hypothetically large problems with more numerous attached processors. The resulting efficiencies for the first-order method are depicted in Fig. 1 *vs* the number of mesh cells per direction, I, where we set $P = n(n+2)/2$, the largest number of independent discrete ordinates in an S_n quadrature, thus the largest speedup factor, for various values of n. These plots indicate that for large P corresponding to very large angular quadratures, the Fan-in Fan-out scheme is more efficient for small meshes, but that the situation is reversed as the number of cells per direction increases. As the communication latency, τ_0, gets smaller, as indeed is the case for the more recent Paragon multiprocessor,[1] the value of I at which the efficiency curves cross decreases. Incidently this behavior justifies using the Fan-in Fan-out scheme on the older iPSC hypercube models which have an even higher communication latency than the iPSC/860. It is evident from Fig. 1 that for n within currently acceptable practical limits, the Bucket scheme is more efficient than the Fan-in Fan-out scheme, even for relatively coarse meshes.

References

1. Thomas H. Dunigan, "Communication Performance of the Intel Touchstone DELTA Mesh," ORNL/TM-11983, Oak Ridge National Laboratory, Oak Ridge, Tennessee, 1992.

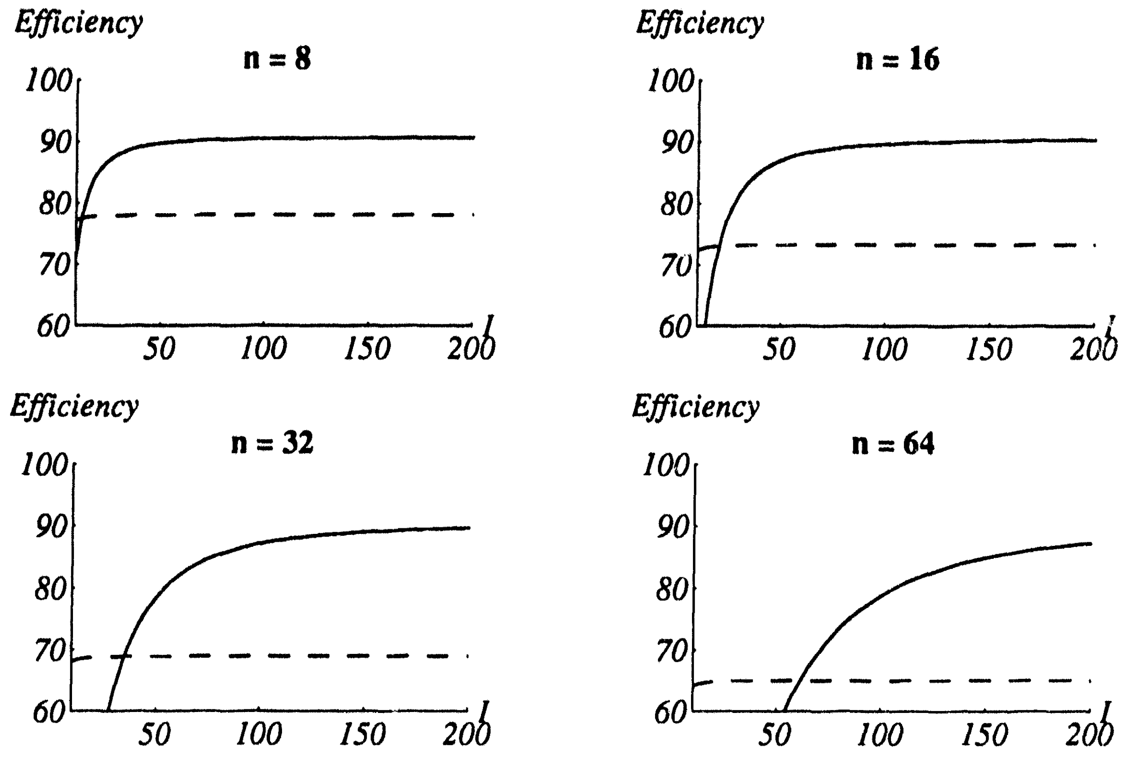
2. Robert A. van de Geijn, "Global Combine Operations" LAPACK Working Note 29, Technical Report CS-91-129, University of Tennessee, Knoxville, Tennessee, 1991.

3. Y. Y. Azmy, "General Order Nodal Transport Methods and Application to Parallel Computing," *Transport Theory and Statistical Physics,* **22,** 359 (1993).

Figure Captions

1. Parallel efficiency for the first-order *P-GONT* code with the Bucket (solid) and the Fan-in Fan-out (dashed) schemes *vs* the number of computational cells per direction on $n(n+2)/2$ processors and various values of n.

$P = n\,(n+2)/2$


Efficiency
n = 8
Efficiency
n = 16
Efficiency
n = 32
Efficiency
n = 64
100
90
80
70
60
50
100
150
200
l

DATE
FILMED
1 / 5 / 94
END